Title:

Efficient Feature-Based Contour Extraction

Author(s):

James R. Gattiker

Submitted to:

http://lib-www.lanl.gov/cgi-bin/getfile?00796675.pdf

Efficient Feature-Based Contour Extraction

James R. Gattiker
Los Alamos National Laboratory
gatt@lanl.gov

Abstract

Extraction of contours in binary images is an important element of object recognition. This paper discusses a more efficient approach to contour representation and generation. This approach defines a bounding polygon as defined by its vertices rather than by all enclosing pixels, which in itself is an effective representation. These corners can be identified by convolution of the image with a 3x3 filter. When these corners are organized by their connecting orientation, identified by the convolution, and type, inside or outside, connectivity characteristics can be articulated to highly constrain the task of sorting the vertices into ordered boundary lists. The search for the next bounding polygon vertex is reduced to a one dimensional minimum distance search rather than the standard, more intensive two dimensional nearest Euclidean neighbor search.

1 Introduction

This paper describes a method to decompose the contour extraction from binary images into two stages. The first stage is a parallelizable identification of a reduced set of contour features. The second stage is combinatorial contour construction, that is greatly reduced in complexity from simple contour following approaches. This new algorithm can greatly reduce the computational burden of the problem compared to previous algorithms for contour extraction, while at the same time allowing the parallelization of the algorithm.

The most basic algorithm for contour extraction uses an automaton to traverse the contour pixel by pixel, or a more complex version involving multiple partial lists build during a raster-scan of the image [1]. Although this approach is intuitively clear, it does not take advantage of the parallel nature of the task, nor of the useful properties of the contours. The *chain-coding* method of representation uses the observation that contour boundary pixels can be extracted using a convolution-style operation[2], and contours can be generated using the knowledge that these edge pixels need to be arranged so they are in a list according to connectivity [3]. Other results explore the concept of using contour features with logical completeness to represent bi-level images [4] [5], naming these features *transition points*, developed for purposes of lossless encoding rather than efficient contour extraction.

The method presented in this paper goes beyond previous results by distilling a set of high-level features that not only represent the image contours without loss of information, but have the further property that the logical extraction of contours is efficient through inherent constraints of the proposed feature set.

Section 2 presents the details of a simple example that illustrates the feature-based contour generation concepts. Section 3 fills in details that extend the concept into generality. Section 4 discusses expected performance enhancements with this method.

2 Contour Generation Method Concepts and Example

The contour extraction algorithm is organized into two algorithms that operate in sequence: feature extraction, and contour generation.

2.1 Feature Extraction

Figure 1 shows an example object from which contour features are to be extracted, with features explicitly labelled. These features are the corner pixels in the object, both interior and exterior. The corners are distinctly orientated, so there are four types of outside corners that will be referred to as O1,O2,O3,O4, and there are four corresponding inside corners, I1,I2,I3,I4. The figure shows the method's labeling of outside and inside corners in the object, which can be accomplished in a number of ways. The most straightforward is through the convolution over the image with a 3x3 matrix whose result is to produce a numeric code for each of the 8 types of features. For example, convolve with the matrix

$$\begin{bmatrix} 1 & 2 & 4 \\ 8 & 16 & 32 \\ 64 & 128 & 256 \end{bmatrix} \tag{1}$$

Figure 1: Example image with contour features labelled

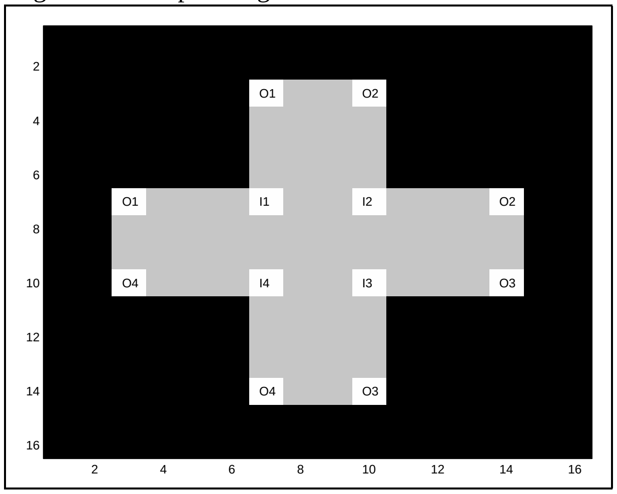


Table 1: List of Figure 1's extracted points by feature type

Feature Type	Member Pixel Coordinates
O1	(7,3) , (3,7)
O2	(10,3) , (14,7)
O3	(14,10) , (10,14)
O4	(3,10) , (7,14)
I1	(7,7)
I2	(10,7)
I3	(10,10)
I4	(7,10)

which results in the numeric results:
$O1 = 27, O2 = 216, O3 = 432, O4 = 54$
$I1 = 255, I2 = 207, I3 = 510, I4 = 447$.

These features are then extracted into unordered lists by type. The lists of points extracted from the shape in Figure 1 are collected in Table 1. We presume throughout that there are no foreground pixels on the edge of the image, which can be assured by padding with a border of background.

The number of contour features is a subset of the whole contour, and will be less than or, strictly, equal to the number of pixel edges in the contour. In typical cases where there are horizontal and vertical runs in the contour, the number of extracted contour features will be significantly smaller than the number of pixel edges in the contour.

These extracted lists represent in themselves a highly compressed complete version of the original image. They may also be used to efficiently generate the contours, as illustrated in the next section.

2.2 Contour Generation

Given the contour feature lists as defined above, the algorithm for the contour generation is a *highly and uniquely constrained* combination of these lists. Once the lists are generated properly, the algorithm is straightforward: Take the starting point for serial contour generation to be the first element of the O1 list, although this choice is arbitrary. The next point on the list, given that we are currently on an O1 point, must come from either the O2 or the I1 lists. Further, the next point will be the closest point on these lists in a specific direction along one dimension: from an O1 point, the next point on the contour must be to the right, and it will be on the same horizontal line. [1]

Table 2: Constraints for contour generation from the contour features

Current Contour Feature Type	Next Contour Feature Type	Dimension and Direction of Next Point
O1,I2	O2,I1	horizontal, right
O2,I3	O3,I2	vertical, down
O3,I4	O4,I3	horizontal, left
O4,I1	O1,I4	vertical, up

The search through the feature lists for the neighboring point on the contour is thus constrained to 2 of the 8 lists of feature points. For each next corner search, the algorithm has two steps: *find exact 1-D match* along the relevant dimension, then *find nearest 1-D distance* along the remaining dimension. This greatly constrains the search for the next contour feature on the contour, in comparison to the general *x-y* search problem to locate the closest pixel in two dimensions. Table 2 shows the complete set of constraints.

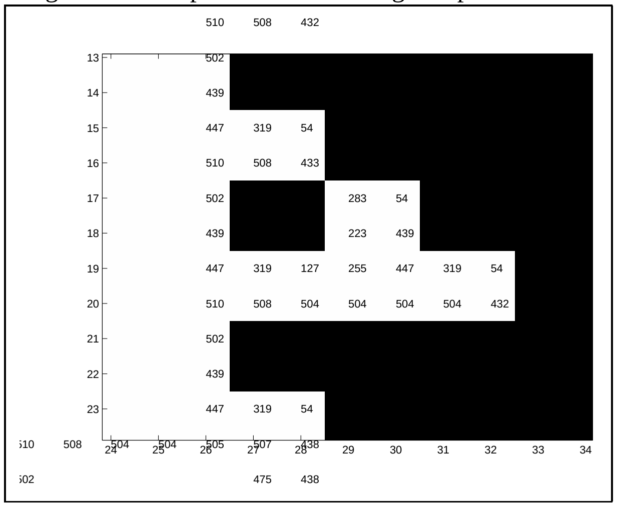
The simplest algorithm for producing a sequential list of contour features is to successively find the closest contour feature according to the above constraints, removing that feature from the list as it is generated. This algorithm can obviously be parallelized, given the observation that every contour stage is independent of the last. Contour segments can be built independently, and the segments constructed iteratively, joining contour corner features to make contour line segments, joining these to make contour corner segments (two line segments at right angles), joining these to make contour 4-line segments and so on. Eventually these contour segments will be joined to make a complete contour.

3 General Contour Extraction

The previous section chose an illustrative example for explanatory purposes, but it does not cover all of the possible configurations leading to inner and outer corners. For example, there are the issues of directly neighboring corners

[1]This description presumes one of two symmetric sets of constraints, namely clockwise exterior border generation.

Figure 2: Example of values in diagonal pixel contact



and corners in diagonal contact. This section will present two alternative approaches to making the presented concept general. The first approach, the double resolution method, enhances the method only slightly to deal with pixel corner contact, but has the disadvantage of doubling the computation of the image convolution/scanning stage. The second approach, the direct corner extraction method, does not require change of the input image, but introduces complexity in the feature labeling and extraction stage. Although these two approaches are equivalent in the abstract the latter may be preferred, because the feature extraction stage, which will usually dominate the total computation time, is relatively smaller.

3.1 Double Resolution Method

The simplest complete method is to double the resolution of the input image, splitting each pixel in the original image into four pixels in the new image. This eliminates neighboring pixel features which cause additional complexity in feature types and contour building, by what can be seen as adding buffer pixels between each feature. The process, from feature extraction through contour generation, can be executed with only minor additions to the feature value lists demonstrated above.

We will assume diagonal contact, i.e. pixels joined only at a single corner point, does not represent connectivity of the foreground object, and conversely does represent connectivity of the background. The converse formulation is also possible.

To complete the list of the above convolution filter's produced values of interest, we must also take into account the additional values generated by diagonal contact. Examples are shown in Figure 2. The four possible diagonal selections add 4 new outside corner values, resulting in the complete feature value list shown in Table 3.

Table 3: Correspondence between pixel value and feature type for the double resolution method

Pixel Value	Feature Type
27,283	O1
54,118	O2
432,433	O3
216,220	O4
255	I1
447	I2
510	I3
507	I4

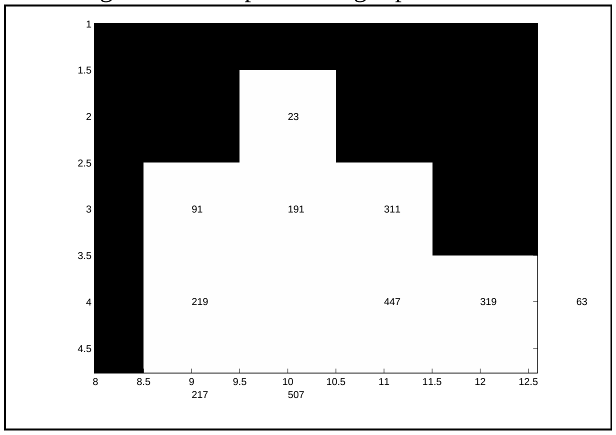
The algorithm using these values and the constraints described in the example section above are sufficient to extract contours from any shape. The next section discusses an alternative algorithm to complete contours generally without change of the original image.

3.2 Direct Extraction of Pixel Corners

Directly moving to the mode of pixel corners during the feature extraction phase is effectively equivalent to the double resolution approach, although the complexity is shifted from the increased computation in convolution, i.e., four times the number of pixels, to the number of features required, trading off complexity for computation.

3.2.1 Single Pixel Features

Figure 3: Examples of single-pixel features



The feature extraction process here must incorporate a much longer list of possibilities. For example, Figure 3 shows new values associated with single-pixel sized features. A significant change here is that the pixels of, e.g., type 23 and 191 in the example, cause additions to two lists, since they have more than one salient feature, for example type 23 represents two outside corners, and type 191 two inside corners.

3.2.2 Closure Asumptions

There are two possible closure assumptions, mentioned briefly above. One possibility is that pixels touching at a single corner represent contact of the foreground and separation of the background. This analysis will take the opposite assumption, that foreground pixels touching at a single corner are not connected. This choice is arbitrary, and the method can be implemented under either alternative.

3.2.3 Sub-pixel Corner Coordinates

It is necessary to take measures to ensure corners from different pixels do not have the same recorded location. The corners from pixel type 191 (see Figure 3) cannot both be listed as the pixel center without introducing combinatorial ambiguity. If the pixel corners are moved to 1/2 native pixel grid spacing (at ± 0.5 of the pixel coordinates), corner positions from pixels touching at a single point will be identical, also causing ambiguity in the combinatorial stage. The solution is to take the corners of a pixel to be inside a pixel, so that for example pixel type 23 corresponds to a I1 feature at (-0.25,-0.25) to the current pixel center, and an I2 feature at (+0.25,-0.25) to the current pixel center. A remapping could instead maintain integer values by simply multiplying the fractional value by four, avoiding the overhead of floating point computation.

3.2.4 Feature Generation

It is in this operation that the overall complexity is increased from the introductory example. Instead of one feature per pixel, we now have a maximum of 4 features per pixel, in the cases of a single pixel (four outside corners) or a single pixel sized cross (four inside corners). It is expected that the average number of features per contour pixel will still allow for significant complexity savings versus methods which maintain all exterior edge pixels, since all horizontal and vertical runs are inherently eliminated. This savings compounds the inherent savings of the constrained combination.

The exhaustive list of pixel values and their associated lists are shown in Table 4. This list includes rotational symmetries expected from the symmetric grid. The convolution algorithm for feature extraction of generating these values then keying to list membership can be reduced to a set of more efficient boolean operations for efficiency.

Table 4: Correspondence between pixel value and feature type for the direct method

Pixel Value	Feature Type
27,31,91,95,283,287,347,351	O1
54,55,118,119,310,311,374,375	O2
432,433,436,437,496,497,500,501	O3
216,217,220,221,472,473,476,477	O4
182,183,246,247,248,249,252,253,255	I1
155,159,411,415,440,441,444,445,447	I2
62,126,218,222,318,382,474,478,510	I3
59,123,315,379,434,435,498,499,507	I4
18,19,22,23,82,83,86,87,274,275, 278,279,338,339,342,343	O1,O2
48,49,52,53,112,113,116,117,304, 305,308,309,368,369,372,373	O2,O3
144,145,148,149,208,209,212,213, 400,401,404,405,464,465,468,469	O3,O4
24,25,28,29,88,89,92,93,280,281, 284,285,344,345,348,349	O4,O1
184,185,188,189,191	I1,I2
154,158,410,414,446	I2,I3
58,122,314,378,506	I3,I4
178,179,242,243,251	I4,I1
254	I1,I3
443	I2,I4
26,30,90,94,282,286,346,350	O1,I3
50,51,114,115,306,307,370,371	O2,I4
176,177,180,181,240,241,244,245	O3,I1
152,153,156,157,408,409,412,413	O4,I2
190	I1,I2,I3
442	I2,I3,I4
187	I1,I2,I4
250	I1,I3,I4
16,17,20,21,80,81,84,85,272,273, 276,277,336,337,340,341	O1,O2,O3,O4
186	I1,I2,I3,I4

3.2.5 Comments on Postprocessing Operations

Postprocessing can determine properties of interest, such as nesting of contours. Extending this to a multi-level image, regions can be annotated with a region type that they enclose. Also, it may be of interest to modify the contours from the pixel boundaries, which lie on the grid of the integers ± 0.25 in the algorithm defined above. Arbitrary remapping of these values can be performed easily, by maintaining and using of the corner *type*. Description of the contours may not require the distinction between inside and outside corners is not necessary, only an indication of the direction from the center of the pixel to the corner is relevant.

This postprocessing can also remap the contour from the quarter- to the half-integer grid, thereby preserving expected notions of area enclosed of one pixel equals one unit, or moving the contour back to integer pixel center values, thereby preserving pixel locations. The nature of the postprocessing naturally depends on the specific nature of further use to be made of the contour.

3.3 Extracting Multiple Contours

If the lists are not empty when the contour is completed, there are additional contours in the image. The methods discussed apply equivalently to interior or exterior contours. Note that if there are remaining contours, there will always be either an outside corner or its corresponding inside corner, so it suffices to check for whether these corresponding lists (e.g. O1 and I1) have entries.

4 Complexity Comparison for Performance Evaluation

Performance assessment is very problem dependent, so performance comparison will be treated apart from specific implementations or images first, with some examples supplied later. In image operations, asymptotic results using order of magnitude are not indicative of perfomance; the number of operations are in reality bounded by real image size, and so the magnitude of the multipliers can be more significant than the strict order of complexity of the operations. In order to abstract performance of this new method compared to other methods, we will consider four basic operations for contour extraction algorithms:

1. Image convolution, for extraction of boundary pixels or corner features, constant $C + M_1 \times n$;
2. 2-D search for nearest neighbor, $M_2 \times n \log^2 n$;
3. 1-D search for nearest neighbor, $M_3 \times n \log n$;
4. local image search for next neighbor boundary pixel, $M_4 \times n$.

where n is the number of boundary features (all contour edges or corner features, as appropriate). The methods we will consider are

(A) automaton contour following, complexity T_4, where n is number of all border pixels;
(B) contour construction from extraction of all borders, complexity $T_1 + T_2$ where n is the number of all border pixels;
(C) the described method of contour construction from corner features, complexity $T_1 + T_3$ where n is the number of corner features.

where T_i is the time for operation i above. Typically, the order multiplier for operation 1, M_1 will be extremely small, simply the time to read out the features of interest. The M_2 and M_3 are the most significant, they are searches of lists, and can be expected to be comparable. M_4 (indexing into image) can be expected to be larger than M_2 and M_3 (accessing a list), but in the same order of magnitude.

Comparing the new method C to method A, we can observe there is added a constant and a very small term scaling with the number of features added for operation 1, but as described M_1 is very small so this component is not considered to be significant for possible image sizes, and the constant C corresponding to convolution will dominate this term. Besides this approximately constant added term, the significant pixel-dependent terms are $O(n_b)$ compared to $O(n_c \log n_c)$, where we must now consider the difference between the n_b, the number of border pixels, and n_c, the number of corner feature pixels.

Comparing methd C to method B, we observe a very similar component for the convolution term, although we are dealing with n_b for method B and n_c for method C. There is a significant difference in the construction phase, both because n_b is greater than n_c, and because of the greater number of operations for 2-D search. The final comparison is $O(n_b \log^2 n_b)$ for method B and $O(n_c \log n_c)$ for method C.

These comparisons critically come down to the magnitude of the ratio $\frac{n_c}{n_b} \in [0, 1]$. As this ratio gets smaller, the advantage of the new method gains. Two images are shown in Figs. 4 and 5. The former is a person walking, segemented based on motion. The ratio $\frac{n_c}{n_b}$ is 0.53. The second is a random image, intended to show a worst-case scenario, The ratio is 0.78. This indicates a favorable position for the new method since this ratio is typically significantly less than one.

Regarding the possible parallel implementation of the algorithms, method A is generally not parallelizable, and methods B and C are parallelizable in a similar style. The advantages of our proposed method remain intact.

Figure 4: Example of contour feature ratio, motion segmented person walking binary image. The ratio of contour corner features to all edge features is 0.53.

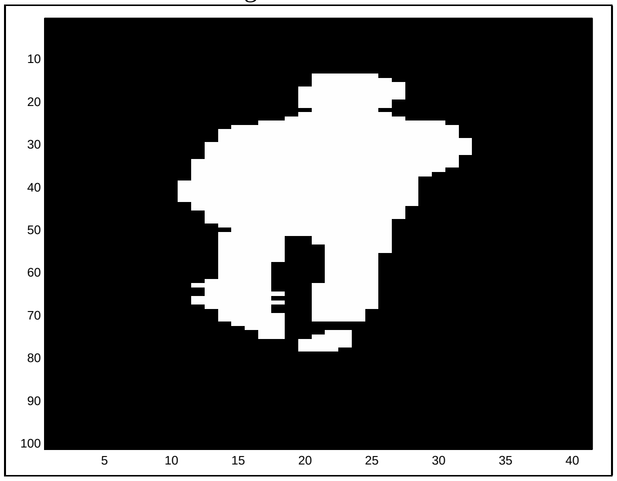
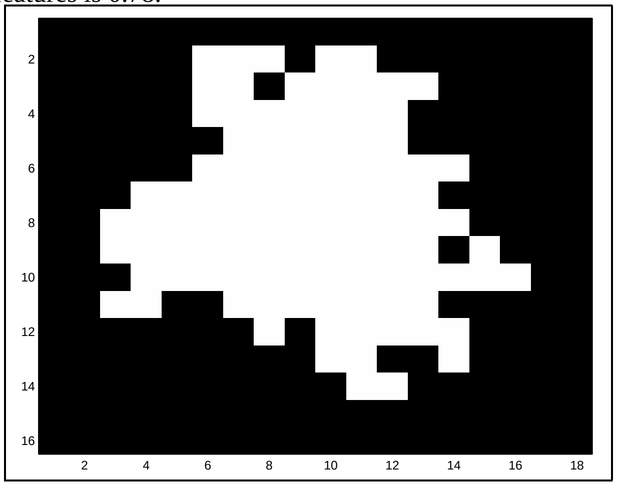


Figure 5: Example of contour feature ratio, random image build under the driteria of randomly adding pixels along the boundary of single starting point. Under the closure assumtions chosen, this shape has three contours, two of which are interior. The ratio of contour corner features to all edge features is 0.78.



5 Conclusions

A new algorithm for efficiently extracting contours was presented. The two stages, *corner feature extraction* and *contour generation*, trade off complexity with computation. The corner feature extraction is an inherently highly parallel operation, generating corner features from an examination of the 3×3 neighborhood centered on each pixel, resulting in a set of corner points annotated with feature types. The contour generation from this representation of the object is a combinatorial optimization problem, with a number of new constraints introduced which greatly constrain the search, in particular avoiding assessment of two-dimensional distance. Although parallel algorithm implementation has not been explored in this paper, this combinatorial problem appears to be decomposable.

This method can potentially be extended from horizontal/vertical connectivity to direct connectivity over diagonal runs. However, the constraints that ease the combinatorial stage of the presented algorithm, specifically the reduction in search dimension from two to one, would be replaced with more challenging $\Delta x = \Delta y$ rules. It is likely that postprocessing the contour to recover diagonals, if desired, is a preferable approach.

References

[1] R.C.Gonzales, R.E.Woods, *Digital Image Processing*, Addison-Wesley, 1992.

[2] H. Freeman, “On the encoding of arbitrary geometric configurations”, IRE Trans. Electron. Comput., col.C-10, pp.260-268, June 1961.

[3] B.R.Schlei, L.Prasad, “A parallel algorithm for dilated contour extraction from bilevel images”, Los Alamos National Laboratory report LA-UR-00-0309.

[4] A.J. Pinho, “A method for encoding region boundaries based on transition points”, Image and Vision Computing 16, 1998, pp.213-218.

[5] A.J. Pinho, “A JBIG-Based Approach to the Encoding of Contour Maps”, IEEE Transactions on Image Processing”, vol.9, no.5, May 2000, pp.936-941.